GENERALIZED DATA STRUCTURES FOR DATA TRANSLATION*

by

Robert W. Taylor

COINS Technical Report 74A-3

September 1974

Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01002

*The research discussed in this paper was partially supported by a grant from the National Science Foundation, GJ41829. The paper was presented at the Third Texas Conference on Computing Systems in Austin, Texas, on November 7 and 8, 1974.

GENERALIZED DATA STRUCTURES FOR DATA TRANSLATION*

by

Robert W. Taylor
Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01002

ABSTRACT

In order for a data translator to deal coherently with a wide variety of data structures as they appear in database management systems, there is a need for a powerful model of data which is simultaneously general and based on as few primitive notions as possible. The DIAM model [1,2,3] is such a model. However, certain structures common to database management systems are not easily modelled in DIAM as originally defined. The paper proposes several extensions to handle these cases, while, at the same time, not extending the basic idea of a few primitive constructs.

The paper also contains a few examples of the use of this model in a data translation situation.

1. Introduction

Data translation can be informally defined as the process which takes data as created by a program and computer system and makes it usable to another program running on a possibly different computer system. Data translation has been the subject of increasing interest over the past few years [4-7]; techniques and methodologies developed in data translators can be of use in a variety of situations:

1) "One-shot" data translation can help alleviate the difficulties of moving files from one computer to another or in preparing files for integration into a database environment.

2) "On the fly" data translation facilities incorporated into a network operating system can greatly enhance a user's capability for file sharing. In particular, a user process could someday gain access to remote data bases, created by some other process, and have data instances presented with a structure that the local process would like to see, rather than with the structure where they were created.

3) Data translation methodology will be useful as research proceeds into adaptive databases--databases which change their structures--particularly their storage structures--depending on usage patterns.

Of course, we are far from the above three goals in the general case--they may well be unattainable in full generality. However, progress is being made on a number of special cases, as will be discussed below.

The rest of this paper will discuss research into some of the necessary features of a data translator. In particular, we will focus on the problem of the necessity for a translator internal form into which a wide variety of data structures can be mapped. We argue that this form must be based on a relatively small number of primitives and still be general enough to handle a wide variety of cases. We then discuss the suitability of the DIAM [1,2,3] model, with appropriate extensions, for such a translator internal form. The final section discusses several open problems.

*This research was supported in part by grant GJ 41829 from the National Science Foundation.

2. Translator Internal Forms

Since the earliest days of Input-Output Control Systems, workers dealing with data have realized the desirability of a standard system internal form for data, the form being independent of the peculiarities of the storage device on which it resides. Thus, we see in most operating systems the concept of a "device support routine" and the separation of logical I/O from physical I/O. These concepts allow higher levels of the operating system to view all devices in a uniform manner.

Similarly, workers in the field of data translation have realized the necessity of a translator internal form. All the existing data translators follow the basic architecture shown in figure 1. A reader/parser process is responsible for accessing data instances and reconstructing the standard form from whatever encoding has been used. The rules for doing this are contained in tables which have been derived from a detailed data description of the source file. Then, the translation/restructuring process constructs the corresponding data instance (in translator internal form) for the target file. Finally, the writer process uses the encoding rules of the target file description to encode the target data instance into the form expected by the target system and write it to the storage medium which in turn will be mounted on the target system.

Clearly then, the translator internal form is an important part of this overall architecture. The capabilities of a particular data translator will be limited by the generality of this translator internal form. The form has the following requirements:

1) It should describe the data as fully as possible, both logically and physically.

2) It should be possible to parse and transform uniquely data instances from a wide variety of data structures, as they currently exist, to an appropriate translator internal form.

3) It should be possible to define a variety of operations over data in translator internal form such that data may be written in a wide variety of encodings.

4) The translator internal form should be based on a few primitive notions of data, together with rules of combination for more complex structures.

We discuss each of these requirements in turn.

The first requirement might be restated as follows; "a data translator must be aware of the meaning of every bit that is present in the source file or to be created in the target file. Its view of data is that which is actually present or which is to be created." The translator is, after all, reading real files and creating real files. Thus, for example, while a user's view of data might well follow the relational model, none of the relational models advocate that the implementation be in terms of "flat, sequential files" necessarily. Rather, they allow for a variety of physical structures which can support a relational view, if desired. But a data translator, since it deals with the data that is "really there," must be aware of such things as lengths and character sets of items, the

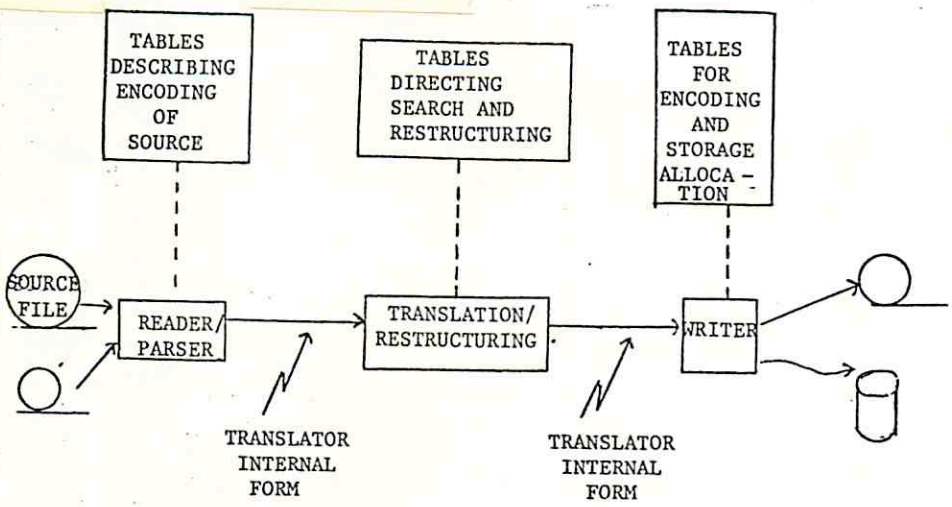


Figure 1: Translator Internal Architecture

structure of pointers, the methods used to delimit one item from another, the algorithm for determining when one entity is considered to be related to another entity, and many other attributes of data which are beyond the scope of this paper, [8].

The point is, however, that while a relational view of data may be entirely appropriate for many classes of database users, it cannot be appropriate for a data translator which deals with a wide variety of source and target media and structures.

To further emphasize this point, consider the concept of the "traversable access path." This is a concept which is implicitly present in seemingly every database system, yet one which is almost never dealt with in a totally explicit manner. Consider the following examples of an access path:

1) The access path leading from one item in a record to the next.

2) The access path leading from one record to the next of the same type in a file.

3) The access path leading from a record of one type to zero or more records of a second type.

Often, these access paths (especially cases 1. and 2) are encoded via contiguity. They are access paths nonetheless and there must be a mechanism in the translator internal form to represent them explicitly. We note in passing that cases 1 and 2 are often not represented implicitly by contiguity in real situations--the indexed sequential file's sequential access path usually follows a mixed strategy, with the access path sometimes represented by contiguity in the address space and sometimes represented by a pointer to an overflow chain; of course, there will be a means of determining which method is used in any particular record instance. The overriding point is that the translator must make the concept of an access path fully explicit and in a standard form. Only then can the problems of a variety of encodings be dealt with.

The second and third requirements follow also from the concept of the translator view of data. If we assume (perhaps optimistically) that the original creator of the data in the source file had associated with it a unique semantics, then it should be possible to transform that data into translator internal form while retaining the semantics. Similarly, when writing translated data, a wide variety of encodings should be possible so that the translator output can be fed to a variety of computer systems.

The fourth requirement should be obvious in the sense that current work in data structures and programming languages, e.g., [9,10] continues to emphasize that need for building complex structures from simple but powerful ones, rather than starting with complex notions to begin with. Only then can one hope, for example, to prove the correctness of a data translation.

3. The DIAM Model

Probably the most complete model of data that has appeared in the literature to date is the Data Independent Accessing Model (DIAM) of Senko, Altman, Astrahan, and Fehder [1,2]. A more tutorial treatment of this model can be found in [3]. We will briefly review the highlights of this model.

The model is broken into four levels--the entity set level, the string level, the encoding level, and the physical device level. The entity set level is a user's view of data as a set of "flat files" of entity descriptions. The entity set level is in many ways similar to the relational view of data and, as argued in section 2, is not the appropriate view of data for a data translator.

The string level is a further specification of the data objects at the entity set level. Specifically, the string level deals with the methods used for grouping data items as specified at the entity set level together with a specification of all access paths both within and among these data items. There are three basic string operations--the grouping operation, the collecting operation, and the aggregating operation.* (See figure 2). The grouping operation specifies an access path from one data item to another; only data items can be grouped. The collection operation specifies an access path among various instances of the same schematic structure--the simplest example would be the access path in a sequential file from one record instance to the "next" record instance. The aggregating operation specifies an access path from an instance of one structure to an instance of a different structure. The aggregate is similar to the DBTG set [12] in some cases.

The encoding level specifies how each object at the string level is encoded into bit patterns, and the physical device level specifies how the bits are allocated over the various media as organized for a partic-

*These were called A-strings, E-strings, and L-strings, respectively, in the original DIAM literature. We will use the terminology of [8].

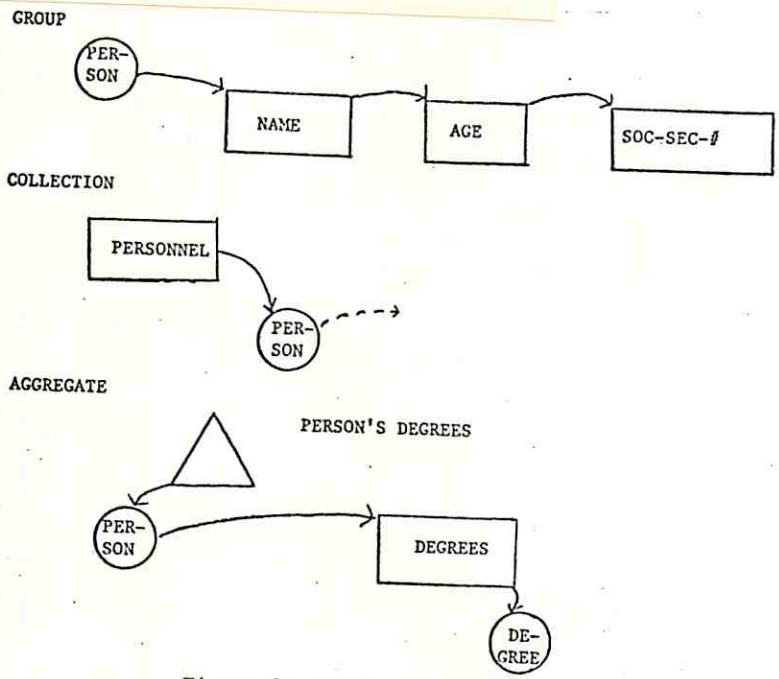


Figure 2: String Level Primitives

ular accessing method. While these two levels of data are important and challenging areas, they are beyond the scope of this paper which will focus primarily on extensions at the string level. The interested reader is referred to [3,11].

We now discuss the attributes of the three string operations in more detail. The grouping operation specifies that an access path must exist from one instance of a data item to an instance of another data item within the group. A group instance is synonymous with an instance of each item in the group (though some of these may have null values which might be represented as a null string when encoded).

The collection operation is specified by a subset selection criterion which specifies when an instance of the structure over which the collection is defined is to participate in the collection. There is also an ordering criterion, which specifies how the various instances within a collection are to be ordered. Finally, it is worth noting that one of the most important subset selection criteria is the "partition by unique value," as shown in Figure 3. In this case, there are as many instances of the collection as there are unique values of the partitioning component. Such a facility will be especially useful in conjunction with the next structure.

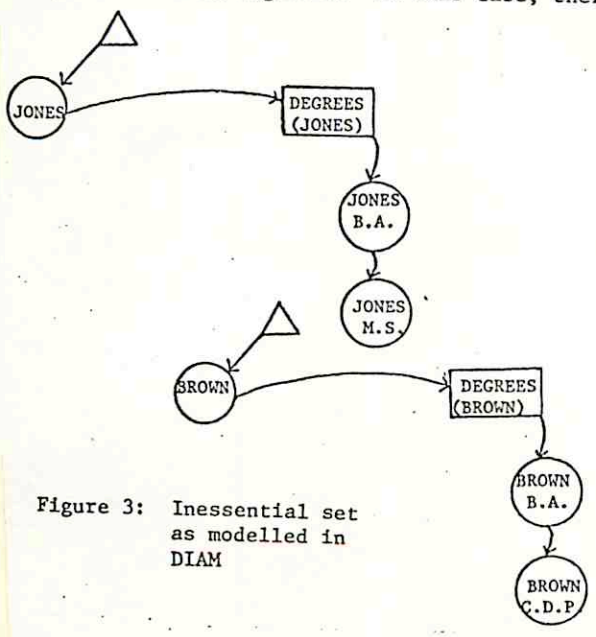


Figure 3: Inessential set as modelled in DIAM

The aggregate relation is specified by naming the components to be aggregated and specifying a match condition--a predicate that must hold whenever the access path exists from an instance of one of the structures to an instance of another.

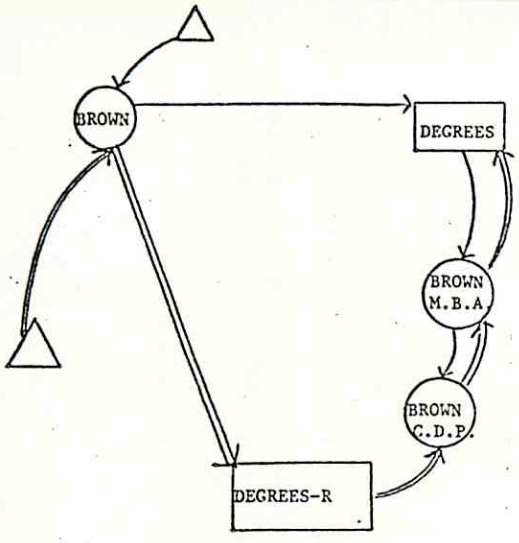
As an example, Figure 3 shows the DIAM model of a DBTG set [12]. The set is modelled as an aggregate defined over an instance of the group PERSON through the collection DEGREES of instances of the group DEGREE such that the DEGREE instances are partitioned into collections by a unique value criterion and the aggregating match condition is over the partitioning criterion. Such a structure corresponds to an inessential set, as defined by Codd [13].

4. DIAM Extensions

The DIAM architecture has been shown [1] to be sufficient, at least in concept, to deal with a variety of data structures and data structure encodings in various database management systems. However, careful examination of a few examples reveals that the DIAM model as originally defined, cannot deal with some structures. We therefore will propose in this section a number of extensions which will allow the same basic constructs to be used in a real data translator.

Consider the example of a DBTG set. It is well known that there are in most existing database management systems a variety of "set modes" used to implement this structure. Among these are chain, chain linked to prior, chain linked to prior and owner, and pointer array. All of these structures imply the existence of four access paths--a path to the "next" member of the set, a path to the "prior" member of the set, a path to the "owner" member of the set, and, of course, a path from the owner to the "first" member of the set. Only two of these are directly embodied in the DIAM L-string concept. Thus, it will be necessary to augment the DIAM concepts to allow the definition of a "prior" path and an "owner" path.

Considering first the prior path (Figure 4) it is clear that the "prior" path is a second collection having the same subset selection criterion and the "reverse" ordering criterion. Thus it should be clear



COLLECTION DEGREES-R
OVER DEGREE
REVERSE OF DEGREE

Figure 4: Extending DIAM for DBTG Sets

that the "prior" path is very much a dependent one--it is only meaningful if the subset selection criterion is the same and the ordering reversed. Thus the DIAM collection statement must incorporate an option stating that one collection is the REVERSE OF or SYMMETRIC TO another, basic collection.

Considering next the "owner" path, (see Figure 5) it is clear that the basic criterion for the existence of this path is that whenever there is a path

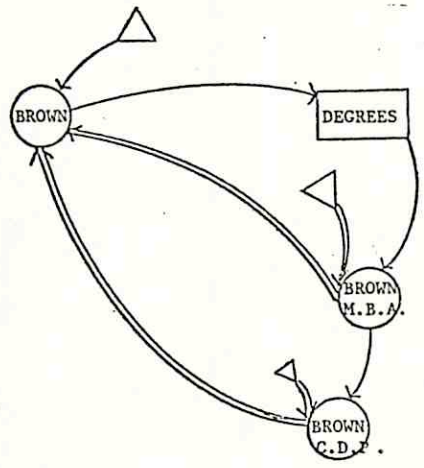


Figure 5: Modelling Owner Pointers in DIAM

from an instance of PERSON, through the DEGREES collection to an instance of DEGREE, then there is a path directly back to the instance of A. Fine. However, this is illegal in DIAM as originally defined. The problem is that the path must be modelled as an aggregate, but as originally defined, it was illegal for a given structure (in this case, BROWN) to participate in more than one instance of a given higher level structure. Thus the multiple instance of DEGREE, all pointing at the same instance of PERSON (i.e., BROWN) is an illegal DIAM structure! However, it is not difficult in this case to declare the structure to be legal since the object pointed at (namely, PERSON) need not point back directly. Hence there is not the problem of handing a variable number of pointers associated with PERSON.

A final example of a necessary generalization is as follows. The aggregate structure of DIAM required that in order for an aggregate to exist, one instance of each of its defining components must exist such that the match condition is satisfied. This rule thereby precluded the concept of, for example, the DBTG empty set as well as the IMS [14] structure where dependent segments can have zero or more occurrences relative to a parent segment. Thus we extend the definition of an aggregate to specify whether a group instance is optional or not in order for the aggregate to exist.

5. The Treatment of Pointers

It should be clear that, with the extensions defined in the previous section, there is within the translator internal form an explicit representation of every access path. Further, these access paths are known to the formation of a particular data instance in the target file.

There remains the problem of specifying the appearance of pointers in the bit stream representing the source or target file. It should be noted that in the original DIAM specifications, this problem was not dealt with in detail--it didn't have to be because the addressing structure was fixed in the DIAM model. A data translator, however, must be able to deal with a variety of device organizations.

As an example, consider the problem of transforming a record structure represented in translator internal form as a list (see Figure 6) into an encoded "indexed"

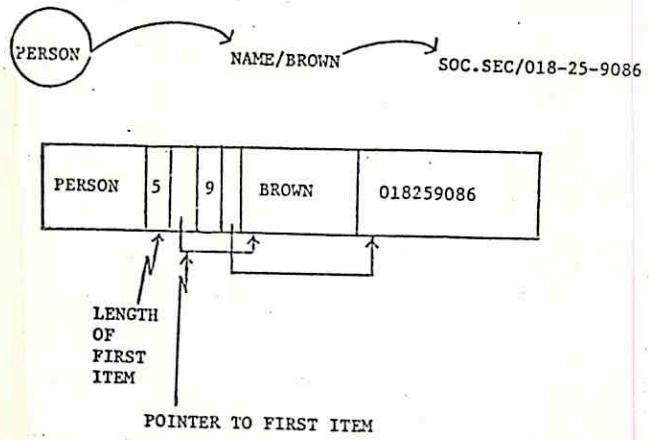


Figure 6: Multiple encodings of items in a record

record structure, where the first part of the record is essentially a vector of pointers locating the various fields of the record. Such a structure is not uncommon in systems that deal with variable-length character strings, such as document systems. Such a transformation was not specifiable in DIAM as originally defined. DIAM contained a number of options whereby one could specify that a pointer was implicitly represented by contiguity of various data, but if a pointer was to remain explicitly represented in the data, there were no facilities for "moving it" to a different place.

Thus it is necessary to define extra operations for moving data items and pointers from one item to another.

The example of Figure 6 is treated as follows (see Figure 7):

1) A collection is defined over a group containing an item into which the pointer will be moved.

2) The pointers in the translator internal form of the group (i.e., list) are treated as a vector and extracted into the collected pointers. Similarly with length attributes.

3) The vector of pointers is represented contiguously and the items in the group are also with

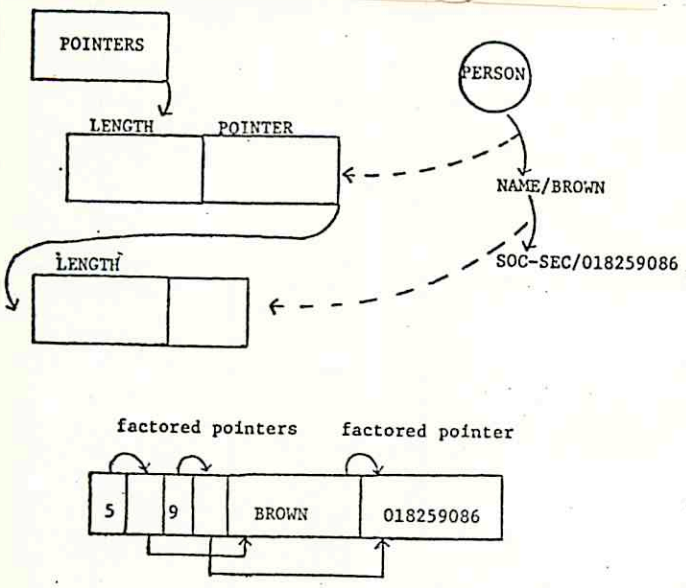


Figure 7: Extended DIAM modelling of pointers embedded in a record structure.

the vector of pointers being contiguous with the first item in the group.

It thus is clear that there must be defined for each pointer that will appear in the encoded data stream an item into which the translator internal form pointers may be extracted. This is necessary because each pointer will eventually be bound to a particular addressing schema and its bits will relate to parts of a device (first 8 bits for cylinder, next 4 bits for track, etc.) and thus the semantics of the pointer must itself be described.

We also note that the notion of extraction of a vector of values and embedding of those values in a collection is a necessary feature for attributes of data other than traversable access paths encoded via pointers. In particular, to distinguish "namelist I/O" from formatted I/O in FORTRAN, the rules for determining how to locate the name of the item in itself constitute the encoding characteristics of a separate item--the name.

6. Conclusions and Final Comments

One of the inescapable conclusions one draws from the discussions above is that data translations and data models for translation are complicated. They are more suited to intra- and inter-computer communication than to man-computer communication. In the longer term, the details of data modelling will not be the concern of many, if any, users, though users will assuredly use parameters in picking their storage organizations. This is as it should be. However, the development of data translation capabilities and the accompanying data models will hasten the day when the computer can help us with our data organizations rather than tending to lock us to a very machine-dependent, non-problem-oriented view as it does today.

It is felt that the extensions to the DIAM model described here enhance the data description capabilities of DIAM to the point where a data translator, based on these concepts, and capable of handling a very wide range of structures could be built. The real question remaining is, how efficient would this translator be. What capabilities would have to be sacrificed when the capability must operate in real-time, as it would a network environment. These questions remain, but in any case, it is unlikely that database designers cognizant of data translation will continue to tie their structures so tightly to programs and machines.

ACKNOWLEDGEMENTS

The author would like to thank R. Frank and D. Smith of the University of Utah, V. Lum of IBM Research, and J. P. Fry of the University of Michigan for many helpful comments.

REFERENCES

1. Senko, M. E., E. B. Altman, M. M. Astrahan, and P. L. Fehder, "Data Structures and Accessing in Data-base Systems," IBM Systems Journal, 12:1, 1973, pp. 30-93.

2. Altman, E.B., M. M. Astrahan, P.L. Fehder, and M. E. Senko, "Specifications in a data independent architectural model," Proceedings of the ACM SIGFIDET Conference, 1972, available from ACM.

3. Mathers, J. O. and R. W. Taylor, "The Data Independent Architectural Model: A Review and Discussion," Report 74A-1, COINS Dept., University of Massachusetts, February, 1974.

4. Fry, J. P., D. C. Smith and R. W. Taylor, "An Approach to Stored-Data Definition and Translation," Proceedings of the ACM SIGFIDET Conference, 1972, available from ACM.

5. Merten, A. G. and J. P. Fry, "A Data Description Language Approach to File Translation," Proceedings of the ACM SIGFIDET Conference, 1974, available from ACM.

6. Ramirez, J. A., N. A. Rin, and N. S. Prywes, "Automatic Generation of Data Conversion Programs Using a Data Description Language," Proceedings of the ACM SIGFIDET Conference, 1974.

7. Housel, B. C., V. Y. Lum and N. Shu, "Architecture to an Interactive Migration System," Proceedings of the ACM SIGFIDET Conference, 1974.

8. CODASYL Stored Data Definition and Translation Task Group, Draft Specifications for a Stored-Data Definition Language, available from author.

9. Hoare, C. A. R., "Proof of Correctness of Data Representations," Acta Informatica, 1, pp. 271-281, 1972.

10. Dijkstra, E. W., "Notes on Structured Programming, Structured Programming, Academic Press, 1972.

11. Smith, D. C. P., and R. Frank, "Concepts for a Physical Access Structure Specification of Data," Proceedings of the ACM National Conference 1974.

12. CODASYL Data Base Task Group, April 1971 Report, available from ACM.

13. Codd, E. F. and C. J. Date, "Interactive Support for Non-Programmers, The Relational and Network Approaches," Proceedings of the ACM SIGFIDET Conference, 1974.

14. IBM Corporation, "Information Management System 360, Version 2, System/Application Design Guide, SH20-0910-3.

BIBLIOGRAPHIC DATA SHEET	1. Report No. NSF-OCA-GJ41829-74A3	2.	3. Recipient's Accession No.
4. Title and Subtitle GENERALIZED DATA STRUCTURES FOR DATA TRANSLATION			5. Report Date date of issue 9/74 6.
7. Author(s) Robert W. Taylor			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Computer and Information Science Dept. Graduate Research Center University of Massachusetts Amherst, MA 01002			10. Project/Task/Work Unit No. 11. Contract/Grant No. GJ41829
12. Sponsoring Organization Name and Address National Science Foundation Office of Computing Activities Washington, D.C. 20550			13. Type of Report & Period Covered 14.

15. Supplementary Notes
Presented at the Third Texas Conference on Computing Systems in Austin, Texas, on November 7 and 8, 1974.

16. Abstracts

In order for a data translator to deal coherently with a wide variety of data structures as they appear in database management systems, there is a need for a powerful model of data which is simultaneously general and based on as few primitive notions as possible. The DIAM model [1, 2, 3] is such a model. However, certain structures common to database management systems are not easily modelled in DIAM as originally defined. The paper proposes several extensions to handle these cases, while, at the same time, not extending the basic idea of a few primitive constructs.

The paper also contains a few examples of the use of this model in a data translation situations.

17. Key Words and Document Analysis. 17a. Descriptors

Data Translation
File Conversion
Data Independent Accessing Model

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release unlimited.	UNCLASSIFIED	7
	20. Security Class (This Page) UNCLASSIFIED	22. Price

BIBLIOGRAPHIC DATA SHEET	1. Report No. NSF-OCA-GJ41829-74A3	2.	3. Recipient's Accession No.

4. Title and Subtitle

GENERALIZED DATA STRUCTURES FOR DATA TRANSLATION

5. Report Date 9/74 date of issue

6.

7. Author(s)

Robert W. Taylor

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Computer and Information Science Dept.
Graduate Research Center
University of Massachusetts
Amherst, MA 01002

10. Project/Task/Work Unit No.

11. Contract/Grant No.

GJ41829

12. Sponsoring Organization Name and Address

National Science Foundation
Office of Computing Activities
Washington, D.C. 20550

13. Type of Report & Period Covered

14.

15. Supplementary Notes

Presented at the Third Texas Conference on Computing Systems in Austin, Texas, on November 7 and 8, 1974.

16. Abstracts

In order for a data translator to deal coherently with a wide variety of data structures as they appear in database management systems, there is a need for a powerful model of data which is simultaneously general and based on as few primitive notions as possible. The DIAM model [1, 2, 3] is such a model. However, certain structures common to database management systems are not easily modelled in DIAM as originally defined. The paper proposes several extensions to handle these cases, while, at the same time, not extending the basic idea of a few primitive constructs.

The paper also contains a few examples of the use of this model in a data translation situations.

17. Key Words and Document Analysis. 17a. Descriptors

Data Translation
File Conversion
Data Independent Accessing Model

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release unlimited.	UNCLASSIFIED	7
	20. Security Class (This Page) UNCLASSIFIED	22. Price